Solving Ising Model Optimization Problems with Restricted Boltzmann Machines on Google Tensor Processing Units (TPUs)

Kaitlyn Chan

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-90
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-90.html

May 10, 2023

Acknowledgement

Thank you to Saavan Patel, Professor Salahuddin, and the rest of the RBM subgroup for their encouragement and guidance as I learned how to research throughout my undergraduate studies up until now. Thank you to my friends and family as well for their support throughout my education!

Solving Ising Model Optimization Problems with Restricted Boltzmann Machines on Google Tensor Processing Units (TPUs)

by Kaitlyn Chan

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Sayeef Salahuddin
Research Advisor

5.8. 2023

(Date)

* * * * * * *

Professor Sophia Shao
Second Reader

5/10/2023

(Date)

Solving Ising Model Optimization Problems with Restricted Boltzmann Machines on Google Tensor Processing Units (TPUs)

Kaitlyn Chan

May 2023

Abstract

The purpose of this project is to demonstrate the capability of the Restricted Boltzmann Machine (RBM) to solve NP-Hard combinatorial optimization problems at scale. It uses the JAX framework to distribute the model across a mesh of interconnected TPU nodes for sampling. The RBM's parallel sampling scheme motivates large scaling across both the batch dimension and problem size to best take advantage of the TPU's architecture for fast and large matrix multiplies. Various sampling algorithms for inference on RBMs are implemented on the TPU and compared to other hardware accelerators. It can currently run up to a 100,000-node MAXCUT problem distributed across 8 TPUv2 cores, which represents one of the largest implementations of such a problem to date.

Table of Contents

1 Introduction

1.1 Background

Combinatorial Optimization problems have applications across a variety of domains, from scheduling logistics to VLSI routing. It is challenging to efficiently solve these problems since their search space scales exponentially with input size. As it is infeasible to exhaustively test all possible solutions, combinatorial optimization problems are NP-Hard, and therefore lack an algorithm to find a solution in polynomial time.

Many NP-Hard and NP-Complete problems can be mapped to the Ising Model, a graph based in statistical physics that models the spin states of magnetic dipoles [1]. It has been of interest to develop different approximation algorithms and hardware systems to efficiently solve the Ising Model problem, and therefore combinatorial optimization problems.

1.2 Motivations and Previous Work

Existing physical accelerators that work on solving combinatorial optimization problems mapped to the Ising Model include quantum computers (DWave 2000Q Computer), optical oscillators that model quantum behavior (Coherent Ising Machines), and hardware accelerators (GPUs, FPGAs, ASICs) that implement novel sampling algorithms.

The purpose of this work is to investigate the Restricted Boltzmann Machine (RBM), a stochastic neural network that maps directly onto the Ising Model, as a comparable computational approach to solving large combinatorial optimization problems. The RBM's parallel sampling scheme motivates large scaling across both the batch dimension and problem size, which takes advantage of the Tensor Processing Unit's (TPU) architecture for fast and large matrix multiplies, as well as distributed programs. We also implement various sampling algorithms suited to the TPU in order to optimize convergence time for the Restricted Boltzmann Machine (RBM).

The largest instances of Ising Model samplers have achieved up to 100,000 nodes with all-to-all connectivity [2, 3], which we demonstrate in our TPU implementation. With further scaling and improvements, we can scale up the RBMs to larger systems on traditional hardware accelerators.

1.3 The Ising Formulation of Max-Cut Problems

The Max-Cut problem is one example of a combinatorial optimization problem. In order to demonstrate the RBM's performance on the TPUs, we benchmark on standardized, complete Max-Cut graphs ranging in size from 1,000 to 100,000 nodes. Solving for the maximum cut of a graph relates to minimizing the energy of the Ising Model in the following way.

Given a graph G with edges E, vertex values $\sigma_i \in \{-1, 1\}$, and edge weights $W_{i,j}$, we want to partition the graph into two sets V^+ (all $\sigma_i = 1$) and V^- (all $\sigma_i = -1$) such that the sum of edge weights between the two sets is maximized. $E(V^+)$ is the set of edges connecting the nodes in V^+, $E(V^-)$ is the set of edges connecting the nodes in V^-, and $\delta(V^+)$ is the set of edges connecting V^+ to V^-.

The derivation of the Max-Cut from the Hamiltonian $H(\sigma)$ of the Ising Model with interactions $J_{i,j}$ between vertices σ_i and σ_j is shown below [4].

$$H(\sigma) = -\sum_{i,j \in E(G)} J_{i,j}\sigma_i\sigma_j \tag{1}$$

$$H(\sigma) = -\sum_{i,j \in E(V^+)} J_{i,j} - \sum_{i,j \in E(V^-)} J_{i,j} + \sum_{i,j \in \delta(V^+)} J_{i,j} \tag{2}$$

$$H(\sigma) = -\sum_{i,j \in E(G)} J_{i,j} + 2\sum_{i,j \in \delta(V^+)} J_{i,j} \tag{3}$$

The first term of (Eq. 3) is independent of the vertex values, so minimizing $H(\sigma)$ is equivalent to minimizing $\sum_{i,j \in \delta(V^+)} J_{i,j}$. By defining the weights $W_{i,j}$ of G as $-J_{i,j}$, the objective is equivalent to maximizing the cut. Defining the cut value $\left|\delta(V^+)\right| \equiv \frac{1}{2}\sum_{i,j \in \delta(V^+)} -J_{i,j}$,

$$H(\sigma) = \sum_{i,j \in E(G)} W_{i,j} - 4\left|\delta(V^+)\right| \tag{4}$$

$$E_{Ising} = \frac{1}{2}H(\sigma) \tag{5}$$

$$\left|\delta(V^+)\right| = -\frac{1}{2}\left(E_{Ising} + \frac{1}{2}\sum_{i,j \in E(G)} W_{i,j}\right) \tag{6}$$

1.4 Mapping the Ising Model to RBMs

The Restricted Boltzmann Machine (RBM) is a two-layer stochastic neural network of binary states. Its structure consists of a layer of visible nodes and a layer of hidden nodes, with connections only existing between the visible and hidden layers (i.e. no intra-layer connections). The RBM is parametrized with weights and biases that represent a target distribution, and Gibbs sampling between layers probabilistically activates states to converge on the minimal energy (highest probability) configuration. With its two-layer structure, Gibbs sampling on the RBM can be massively parallelized as states are updated independently.

The Ising Model can be embedded into the RBM by copying each logical node from the original Ising matrix (A) and replicating it into the hidden and visible layers (B), with a coupling coefficient C forcing corresponding nodes to match states [5].

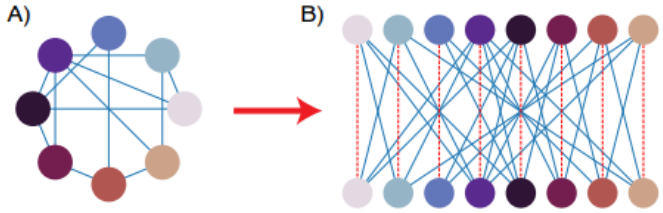

Figure 1: Mapping from the Ising Model to the RBM with twice as many nodes.

Equating the energy functions of the RBM and Ising Model formulations results in the following transformations when mapping an n-node Ising problem onto the RBM with n visible states and n hidden states. σ are bipolar variables in $\{-1, 1\}^n$, and v and h are binary variables in $\{0, 1\}^n$.

$$E_{Ising} = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} J_{i,j}\sigma_i\sigma_j \tag{4}$$

$$E_{RBM} = \sum_{i=1}^{n}\sum_{j=1}^{n} W_{i,j}v_i h_j + \sum_{j=1}^{n} b_j h_j + \sum_{i=1}^{n} c_i v_i \tag{5}$$

To equate the two energies and transform from an Ising problem over ± 1 states to an RBM over binary states, we set $W_{i,j} = W_{j,i} = -4J_{i,j}$, and $b_j = c_i = -2(W\overline{1})$, where $\overline{1}$ is a vector of ones. The mapping from the Ising Model to the RBM informs which data structures are needed to be stored on the TPU. The weight matrix W, biases b and c, along with hidden and visible layers v and h are necessary to perform sampling. Additionally, the adjacency matrix J is also placed on the TPU for calculating cut values from the Ising energy as we sample, though those calculations can also be performed offline on saved sample states.

2 TPU Architecture

2.1 Background

TPUs have been benchmarked for a variety of high performance computing applications, from Ising Model simulations on a single core [6], to training machine learning benchmark MLPerf on a "TPU Pod" of 4096 chips [7]. With its high speed interconnect, dedicated matrix and vector compute units, high bandwidth memory, and distributed computing infrastructure, Google's Tensor Processing Unit (TPU) has an ideal architecture for scaling up parallelized sampling on the RBMs.

Specification	TPUv2 Core	TPUv3 Core
Inter-chip Connect	500 GB/s (4 links)	656 GB/s (4 links)
High Bandwidth Memory	8GB	16GB
Total System Memory	8GB * 8 Cores = 64GB	16GB * 8 Cores = 128GB
Peak TFLOPs/Chip (2 cores)	46	123

Table 1: Compute, memory, and bandwidth increase from TPUv2 to TPUv3 [9].

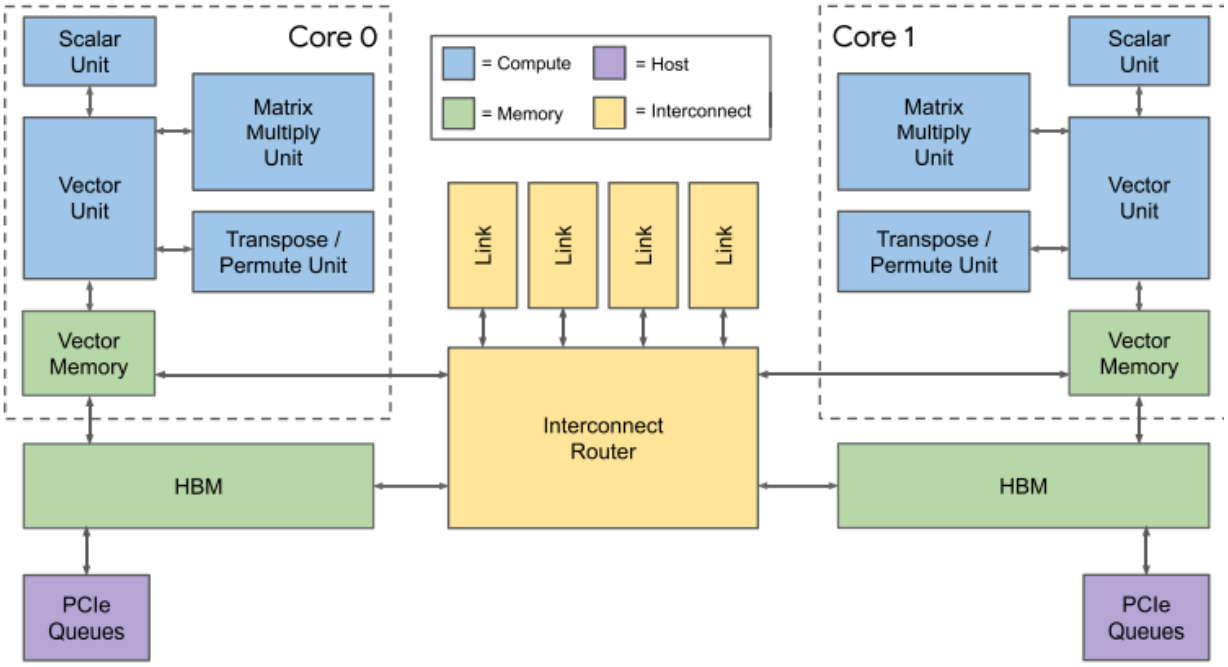

Figure 2: The TPUv2 "Chip" consists of 2 cores, each with their own compute units, HBM, and interconnect bus [10].

All experiments done in this report are performed on 8 TPUv2 cores, with larger Ising problem instances distributed across cores. After problem instances are generated on the TPU host machine, the RBM data structures are transferred to the TPU cores, and all computations remain on device to minimize host-device communication latency.

2.2 Investigation of Compute Precision

The Matrix Multiply Unit (MXU) of each TPUv2 core is a systolic array that performs 128x128 multiply-accumulate operations on inputs rounded down to bfloat16 precision. There have been studies on the capability of low precision data to train machine learning algorithms, and with bfloat16 able to represent the same range as 32-bit floating point (FP32), researchers were able to show that deep learning training in bfloat16 can achieve the same results as FP32 [11].

The benefits of implementing the RBM in bfloat16 precision include the ability to fit larger problem sizes on the TPU, as well as the reduction of data volume passed during inter-core communication. M100000, the 100000-node Ising problem in particular, had to be placed on the TPU devices in bfloat16, as the reduced precision data already required approximately 61GB of memory on the 64GB capacity of the 8 core system. While the binary states of the hidden and visible nodes of the RBM are represented precisely in bfloat16, we ran experiments to check that reduced precision random number generation and acceptance probability calculations for flipping states did not affect the convergence of the RBM to the ground state solution. While Gibbs sampling on reduced precision weights and biases resulted in more strongly quantized cut values, the bfloat16 RBM showed comparable results to its FP32 version when benchmarked on the K2000 Max-Cut problem.

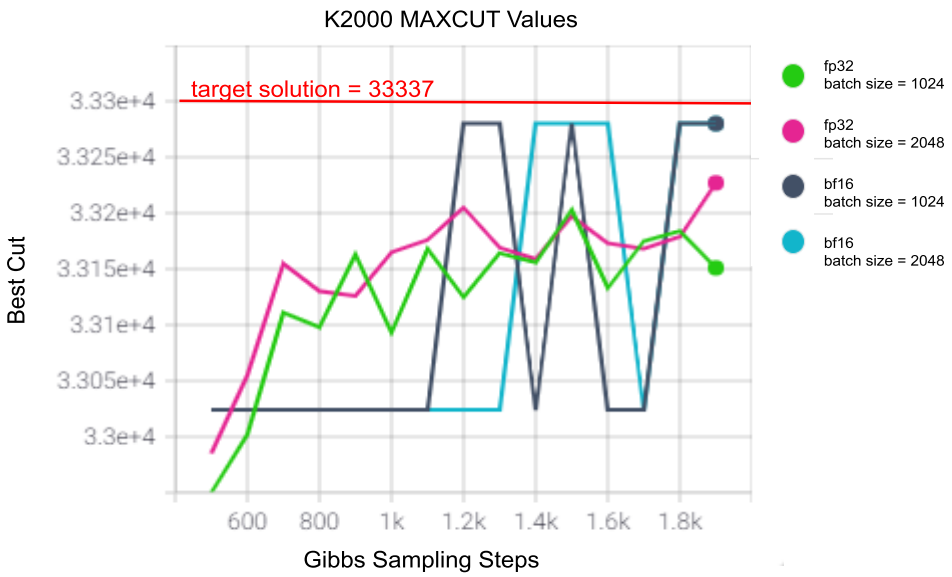

Figure 3: bfloat16 vs FP32 RBM sampling results on K2000, a dense, 2000-node Max-Cut problem.

2.3 Mesh Partitioning and GPU Comparison

When partitioning a large RBM across multiple TPU cores, it is possible to specify various partitioning layouts of the TPU "mesh". For example, for placing a $(100000)^2$-node weight matrix on 8 TPU cores configured in a 4x2 mesh, each core receives a 25000 by 50000 sized slice of the matrix. Mesh scaling on the TPUv2 devices can increase up to 32, 128, 256, and 512 cores total. Interconnect speeds between HBM can be a bottleneck of high performance computing on distributed devices, and partitioning data on a mesh with low perimeter to surface area ratio may minimize intercore communication. We measure Gibbs sampling iteration speeds for the M100000 Ising problem across 1x8, 2x4, 4x2, and 8x1 TPU mesh layouts, but differences in timing results are negligible, leading us to believe that the JAX software framework for distributing matrix-multiplies and other reduction operations hid intercore latency at this small scale TPU mesh. As a result, the rest of the experiments in this report are conducted on a 1x8-core TPU mesh, for flexibility in the data's batch dimension being evenly divisible across the first axis.

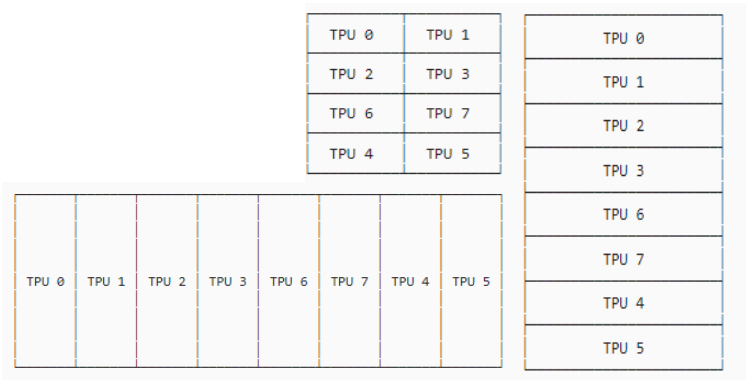

Figure 4: Various TPUv2 8-core mesh partition layouts.

We are also interested in benchmarking the RBM on NVIDIA's A100 80GB GPU for comparison. Our JAX implementation is ported from the TPUs to the GPU with minimal modification, the main difference being that all Ising problem instances were able to fit on the single GPU's 80GB of HBM (still in bfloat16 precision), removing the need for data partitioning and intercore communication. For running 10,000 Gibbs sampling steps on the K2000 Max-Cut problem, the GPU finishes computation of the Parallel Tempering algorithm in 8.5 seconds (green), while the TPUs finishes approximately 1.3x times slower in 14.8 seconds (blue).

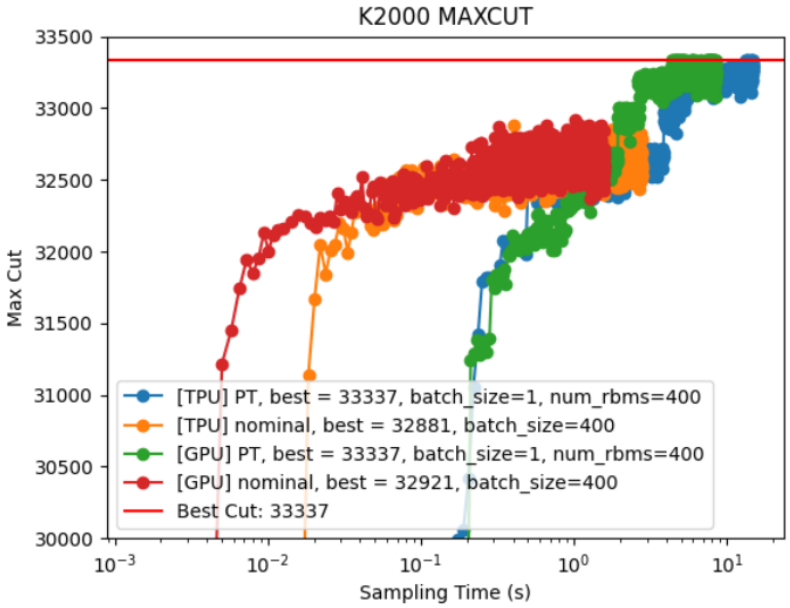

Figure 4: TPU vs GPU RBM sampling results on K2000.

2.4 Searching for Bottlenecks

The first implementation of the RBM on TPU was written in TensorFlow, and using Google Cloud TPU's TensorBoard profiling tool, we took a look at the performance breakdown of the RBM's Gibbs sampling algorithm. For operations computed on the TPU devices, the three matrix-multiplies steps evenly subdivided the bulk of the runtime into thirds, with the Backwards Gibbs pass taking slightly more due to an extra transpose on the weight matrix. However, since the construction of the weight matrix is symmetric, we can eliminate the transpose. The profiling tool also provides the MXU's percentage utilization of the TPU's peak FLOPs, with higher percent utilization indicating faster operations.

Experiments run in TensorFlow on RBMs with varying batch sizes of visible and hidden layers showed runtimes bounded by the data transfer from the CPU host to TPU devices on the PCIe bus. Linear scaling between input data size and runtime required batch sizes of at least 512, so to avoid bottlenecks from PCIe bandwidth limitation, we want to ensure that all input data is fed to the TPU devices before sampling.

Algorithmic Step	XLA Operation	Percent of Device Computation Time	FLOPs % Utilization
Backwards Gibbs	matrix-matrix multiply	35	68
Forwards Gibbs	matrix-matrix multiply	30	79
Ising Energy Calculation	matrix-matrix multiply	28	83
Max-Cut Calculation	vector-scalar multiply-add	7.4	82

Table 2: Top 4 TPU Operations during Gibbs sampling on the RBMs.

3 RBM Implementation

3.1 JAX Framework

While profiling in the TensorFlow framework helps to identify sampling bottlenecks, JAX's documentation and API afford more flexibility for explicit data placement on TPU devices as we begin to scale larger models across distributed devices. At the time we first started working with TensorFlow, the interface architecture was with a Google Compute Engine VM that communicates over gRPC to the TPU's host machine. We started to face memory limitations on the Google Compute Engine VM when generating 100,000-node Ising problems, so we turned to JAX, a NumPy-style API optimized for hardware accelerators that ran directly on the TPU host machine, eliminating the need for a user-side VM. Furthermore, when scaling to a mesh of TPU devices, TensorFlow partitions compiled code from a single global process to each TPU host via RPCs, in comparison to JAX, where each TPU host will compile and run its own copy of the full code, communicating only during collective operations [7].

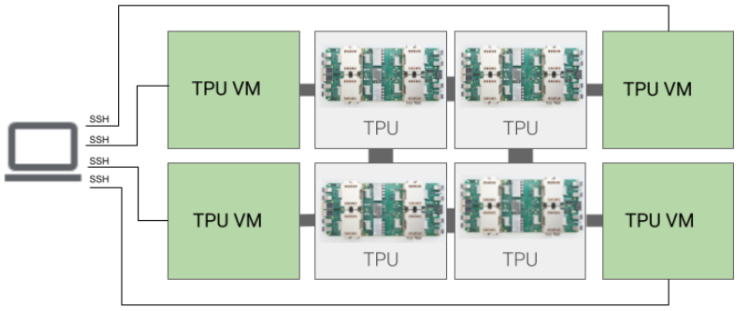

Figure 5: TPU VM System Architecture [12].

JAX, like TensorFlow, is built on top of an XLA compiler that optimizes functions for the hardware accelerator it is running on. It has a few different single-program, multiple-data (SPMD) parallelization methods that we explore. PMAP is one such method that replicates a program across its devices and performs collective operations through the interconnect. Manual sharding of data allows for finer grain control of exactly which data is placed on which device.

The other method, PJIT (now just merged with JIT), consists of an autosharder that automatically partitions the data across specified mesh axes and abstracts all resources as a single global device. The functions are just-in-time compiled and cached on each device, so arguments sizes are fixed and known at compile time.

For smaller Ising problem sizes, it is possible to replicate the entire RBM onto each TPU core and parallelize sampling across the batch dimension, treating each core as an independent sampler. This implementation (using PMAP) is faster than distributing portions of the RBM across cores with PJIT, as computations can be self-contained on each core (Figure 6). However, memory limitations per core necessitate distributing sub-matrices across TPU cores for larger problems.

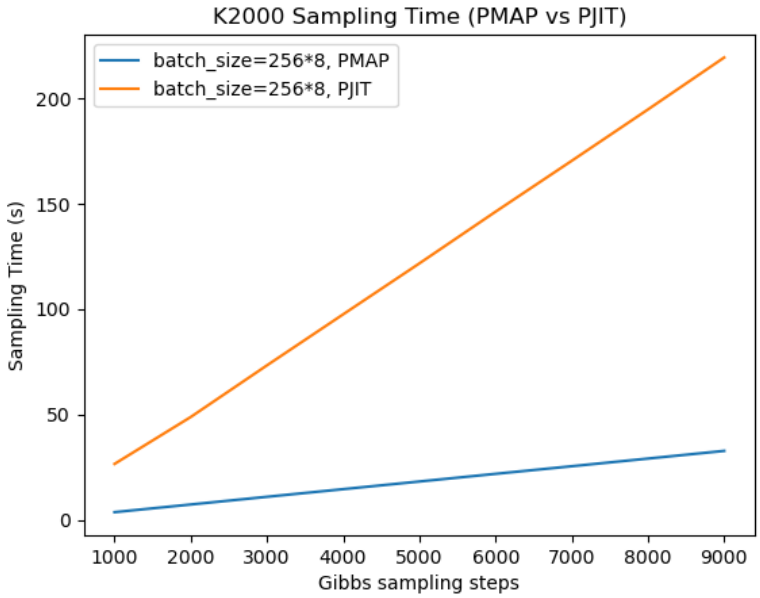

Figure 6: PMAP vs PJIT comparison for RBM sampling on K2000.

3.2 Large Ising Problem Scaling

To benchmark the performance of the RBM on large Ising problems, we generate the M100000, an all-to-all connected 100,000-spin Ising problem with random, continuous weights between -1 and 1, as specified by Goto, Tatsumura, and Dixon in Ref [2]. With approximately 5 billion weights, the RBM construction of the M100000 problem is around 61GB of data in bfloat16 precision, which is too large to fit on a single TPU core with 8GM of HBM.

We first construct the RBM's weights, biases, and adjacency matrix on the TPU Host VM, a process that takes approximately 100 minutes. After setting up the mesh of TPU devices, we then use JAX to partition the RBM across the mesh according to the autosharder.

Sampling code is written PJIT, and operations such as matrix-multiplies and Ising state updates are abstracted across the discrete TPU cores as computations on one large device. By specifying the data axes along which we distribute the inputs (RBM layers and weights), operations can be run in parallel across the devices.

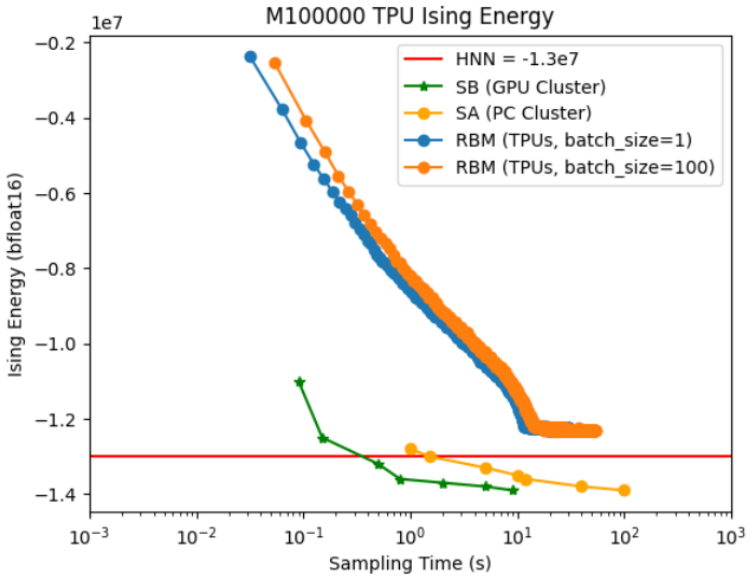

Figure 7: Performance of the RBM distributed across TPU cores compared to other hardware accelerated algorithms in Ref [2].

For M100000, the TPUs take 30 ms per Gibbs sampling step on a visible-state batch size of 1. The scaling strength of the TPU is apparent when increasing the batch dimension of the sampled visible states by a factor of 100 results in a much lower order of magnitude increase in sampling time (~100 ms/sampling step).

In comparison, researchers of a different Ising Model approximation algorithm called Simulated Bifurcation (which also utilizes parallel state updates) run sampling on 8 Nvidia Tesla V100 GPUs, achieving a speed of about 8ms per sampling step [2]. The table below provides more details on the hardware and performance differences of the two algorithms compared to an Ising energy benchmark solved by the Hopfield Neural Network (HNN), a naive local minimum search. The Simulated Bifurcation researchers also provide data point comparisons to their implementation of simulated annealing on a PC Cluster.

Algorithm	Hardware	Interconnect	Total Memory	Total Peak TFLOPs	Time to HNN
Simulated Bifurcation	8 GPU Nvidia Tesla V100-SXM2-16GB	300 GB/s	128 GB	126	8ms/step * 35 steps to HNN solution = **0.28s**
Simulated Annealing	25-node PC Cluster Each Node: 2 Intel Xeon E5-2697-v3	InfiniBand (Datalink speed not specified)	128 GB per node	40	**1s** to HNN solution
RBM	8 TPUv2 Cores	500 GB/s	64 GB	180	30ms/step * 250 steps to **90%** of HNN solution = **7.5s**

Table 3: Hardware and performance differences on the 100,000-node Ising Problem.

3.3 Coupling Parameter Investigation

When transforming from the Ising to RBM formulation, each visible node is mapped to its hidden counterpart. The coupling parameter C is the RBM's "edge weight" ($W_{i,i}$) between corresponding visible/hidden pairs. To disincentive the degenerate cut (simply bisecting the graph across the hidden and visible nodes for a Max-Cut value of zero), we want to effectively set the coupling parameter while keeping the RBM's sampling distribution close to the original model distribution.

Various coupling assignment methods we explore for the +-1 Ising Problem include:

- $C_{i,i} = c$, a fixed constant, determined empirically
- $C_{i,i} = -\alpha \left| \sum_{j=1}^{n} W_{i,j} \right|$, proportional to the absolute value of row sum of weights.
- $C_{i,i} = -\alpha \sum_{j=1}^{n} W_{i,j} \; if \; < \; 0, \; else \; 0$, replacing absolute value above with relu().
- $C_{i,i} = -\alpha \sum_{j=1}^{n} W_{i,j} + \beta$, a linear combination of weight row sums.

The relationship between the RBM's coupling parameter and sampling convergence is not fully understood, but embedding information about the incoming edges to each node empirically improves sampling.

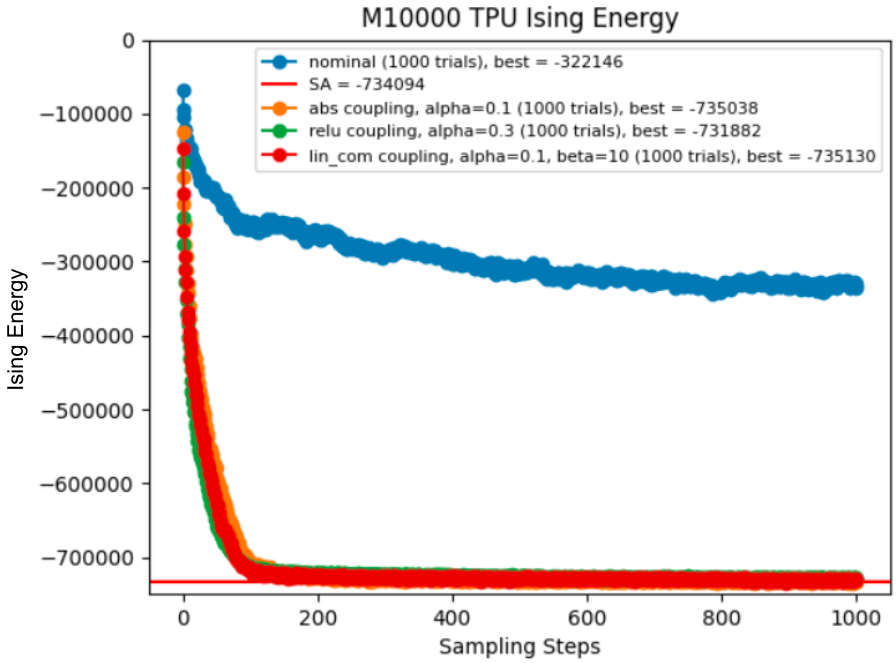

Figure 8: Abs, relu, and lin_com refer to the coupling assignment methods detailed above, while nominal refers to sampling with a fixed coupling coefficient. All other parameters are held the same. SA refers to the Simulated Annealing solution state, which we use as our target.

To understand how various coupling assignment methods affect the sampling distribution, we use a ratio of unnormalized visible state probabilities [13] to characterize how likely the current state is sampled from a distribution close to the model distribution. Using the ratio of marginal probabilities of visible sample states eliminates the need to calculate the partition function Z, which is exponential with the number of nodes (Eq. 6). If the probability ratio is close to 1, then the solution state is ideally sampled from the model distribution, as both probabilities are calculated from the nominal RBM. For investigating sampling distributions of various coupling assignment methods, P_nominal means all couplings are fixed at a constant 2, determined empirically.

$$Z = \sum_{v,h} e^{-E_{RBM}(v,h)} \tag{6}$$

$$p(v) = \frac{1}{Z}\sum_{h} p(v,h) = \frac{1}{Z}\sum_{h} e^{-E_{RBM}(v,h)} \tag{7}$$

$$p(v) = \frac{1}{Z}\prod_{j=1}^{m} e^{b_j v_j} \prod_{i=1}^{n}\left(1 + e^{c_i + \sum_{j=1}^{m} W_{i,j} v_j}\right) \tag{8}$$

We see below that as the Ising problem size increases from 1000 to 10,000 nodes, the nominal coupling assignment method settles to a local minimum, while the other coupling assignments methods are able to track and surpass the simulated annealing solution. For smaller problem sizes, the probability ratios stay closer to 1, though the likelihood of sampling the solution state from the nominal RBM begins to diverge as the problem size increases.

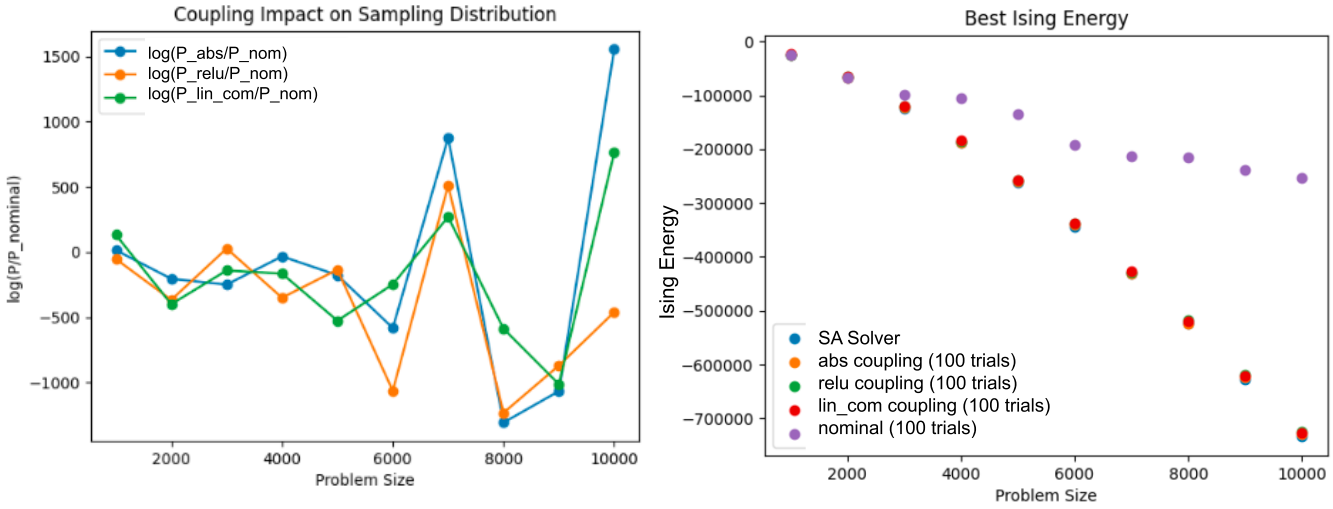

Figure 9: Probability ratios of coupling assignments across Ising problem sizes (Left); Best Ising energy states found by the sampler (Right).

4 Exploration of Sampling Algorithms

4.1 Block Gibbs Sampling

After we set the RBM's coupling terms to be proportional to the absolute value of row sum of weights, basic Gibbs sampling is able to find the solution state with an Ising energy lower than the Simulated Annealing solver for problem sizes M1000 to M10000 (Figure 9, Right).

However, as the number of nodes in the RBM scales up, more Gibbs sampling steps are necessary in order for the Markov chain to converge on the highest probability/lowest energy state. To improve the sampling efficiency of the RBM, we look at other Markov chain Monte Carlo-based algorithms compatible with the TPU's architecture that enhance Gibbs sampling.

4.2 Adaptive Markov Chain Monte Carlo

When sampling with a Markov chain on high dimensional data, the energy landscape of its distribution may have many local minima. Sampling at different temperatures can help the Markov chain reach the ground state, with higher temperatures improving mixing by lowering the barrier for nodes to flip states.

In the Adaptive Markov chain Monte Carlo (MCMC) algorithm, the goal is to have two sampling chains running in parallel [14]. The "slow chain" is sampled at T1 = 1 (or inverse temperature β = 1) and stays close to the model distribution. The "fast chain" is sampled at a range of higher temperatures T2 = 1 to 10 (or β = 0.1 to 1), where the distribution is hopefully smoother and easier to sample from.

In the results shown below, we probabilistically swap the visible states of the slow and fast chains every 50 Gibbs sampling steps according to the Metropolis-Hastings update rule (Eq. 14), where $P_{T1}\left(x_{T2}\right)$ is the joint probability of the fast chain sampled from the slow chain's distribution. Simplifying the acceptance ratio r to its form in Eq. 13, we can reuse the Ising energy values already computed for the Max-Cut calculation at that sampling step. Swapping the two chains is advantageous (more probable) if the fast chain has found a lower energy state, allowing the slow Markov chain to mix while sampling close to the model distribution.

$$P_k(x) = \frac{1}{Z_k} exp\left(-\frac{1}{T_k} E_{Ising}(x)\right) \tag{9}$$

$$r = \frac{P_{T1}\left(x_{T2}\right) P_{T2}\left(x_{T1}\right)}{P_{T1}\left(x_{T1}\right) P_{T2}\left(x_{T2}\right)} \tag{10}$$

$$r = \frac{exp\left(-E_{Ising}\left(x_{T2}\right)/T1\right) exp\left(-E_{Ising}\left(x_{T1}\right)/T2\right)}{exp\left(-E_{Ising}\left(x_{T1}\right))/T1\right) exp\left(-E_{Ising}\left(x_{T2}\right)/T2\right)} \tag{11}$$

$$r = e^{\left[\left(-E_{Ising}\left(x_{T2}\right)+E_{Ising}\left(x_{T1}\right)\right)/T1\right]} e^{\left[\left(-E_{Ising}\left(x_{T1}\right)+E_{Ising}\left(x_{T2}\right)\right)/T2\right]} \tag{12}$$

$$r = exp\left[(\beta 1 - \beta 2)\left(E_{Ising}\left(x_{T1}\right) - E_{Ising}\left(x_{T2}\right)\right)\right] \tag{13}$$

$$P_{swap}\left(x_{T1}, x_{T2}\right) = min(1, r) \tag{14}$$

In Figure 10, we compare Gibbs sampling to Adaptive MCMC on a 10,000-node Ising problem. Successful swaps are marked with green vertical lines. We can see in the Adaptive MCMC plot that the fast chain (orange) samples from a range of tempered RBM distributions (T = 1 up 10 for 25 steps, then back down from 10 to 1 for another 25 steps), and helps reduce the number of sampling steps the slow chain (blue) needs to take to minimize the Ising energy, while ultimately finding a lower energy state compared to Gibbs sampling from a single chain.

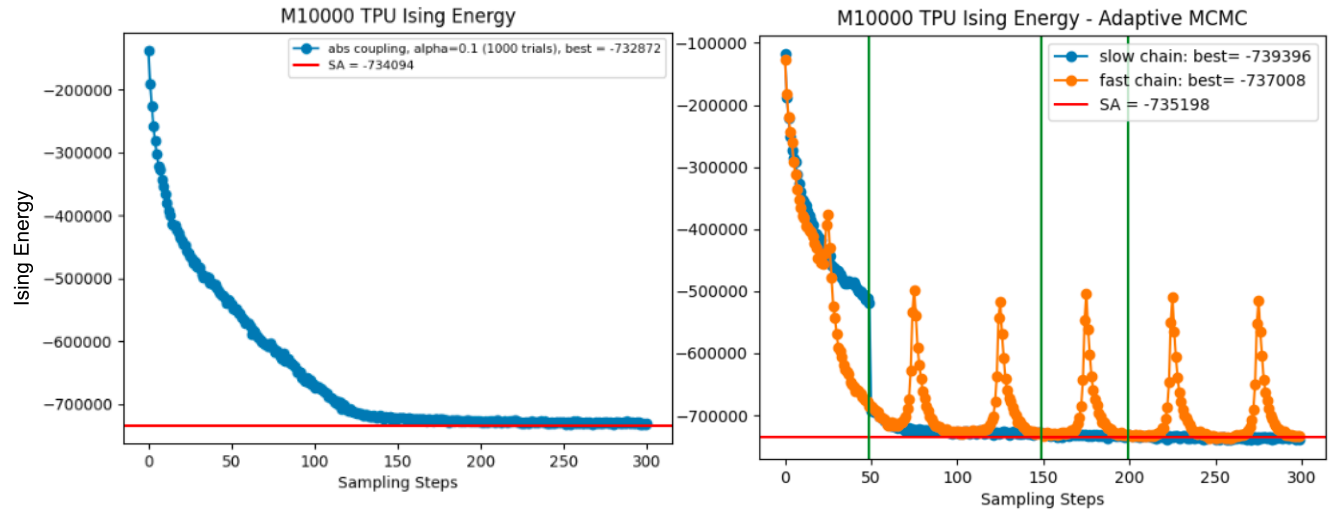

Figure 10: Adaptive MCMC (Right) takes around 75 steps to approach SA solution, while standard Gibbs sampling (Left) takes around 150 steps.

4.3 Parallel Tempering

Parallel tempering is another algorithm that utilizes multiple Markov chains to sample from different probability distributions in order to facilitate better mixing. However, unlike Adaptive MCMC, where one fast chain cycles through various temperature distributions and probabilistically swaps the sample state with one slow chain, parallel tempering uses many more chains sampling at stationary intermediate distributions [15].

While running more than two chains costs more memory resources, the TPU's ability to scale along the batch dimension for parallel operations creates an advantage in allowing even broader exploration of the energy landscape at minimal runtime increase.

For our implementation of the parallel tempering algorithm, we use m number of parallel sampling chains, each running on a temperature constant geometrically spaced between $1 = T_1 < \cdots < T_m = 80$. Every 10 sampling steps, the visible states of neighboring chains

are swapped according to the same criterion as Adaptive MCMC (the Metropolis-Hastings update rule).

The algorithm's swapping portion checks adjacent pairs of chains, alternating between odd and even numbers as the starting index of the pair. More concretely, the algorithm:

1. Performs 10 Gibbs sampling steps in parallel across m chains with distribution P_k for $k = 1 \cdots m$.
2. Calculates swap probabilities for even-leading-index swaps (i.e. {0, 1}, {2, 3}, {4, 5}, ...) in parallel.
    a. Serially reassigns the rows of the visible-state matrix (dimension m-chains by n-visible nodes), due to JAX's requirement for functional array updates [16].
3. Performs another 10 Gibbs sampling steps in parallel across m chains.
4. Calculates swap probabilities for odd-leading-index swaps (i.e. {1, 2}, {3, 4}, …) in parallel.
    a. Serially reassigns the rows of the visible-state matrix.
5. Takes the visible state from the $k = 1$ RBM as the sample from the model distribution.

The method of deterministically alternating between even and odd pairs during the swapping segment of the algorithm has been compared to alternatives such as random selection of the exchange pairs and swapping all potential pairs (not just adjacent neighbors) [17]. Through experiment, the researchers showed that the deterministic even odd (DEO) exchange method yielded the highest "round trip rate" of a sample state traversing the full temperature ladder, so we decided to use this method.

In Figure 11 below, we plot the progression of various sampling algorithms on the 5000-node Ising problem. We began with a nominal RBM with standard Gibbs sampling and a constant coupling parameter (green). Then, coupling adjustments to the RBM (orange) combined with parallel sampling from a range of distributions (blue, red) sped up convergence to the ground state solution.

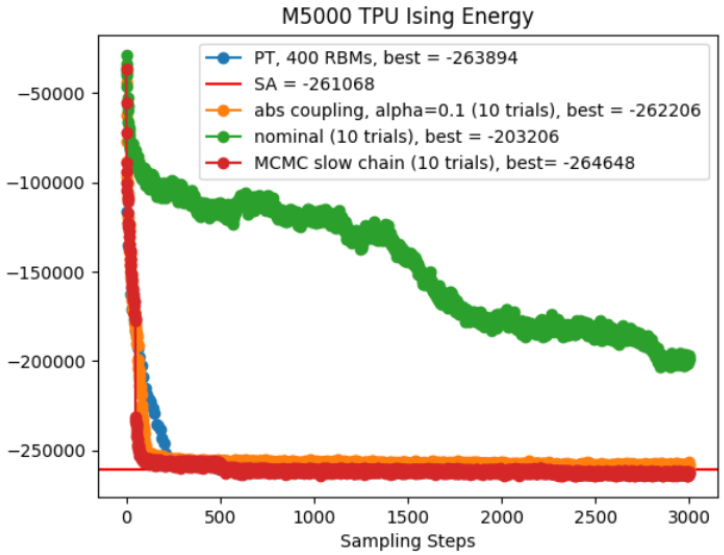

Figure 11: A comparison of various sampling algorithms on the RBM.

Runtime differences on the TPU for the 3,000 Gibbs sampling steps on various algorithms for a 10,000-node Ising problem are shown below. Standard Gibbs sampling (green) has the quickest time per sampling step, as the entire algorithm is parallelized. Parallel tempering (orange) has a longer runtime that is proportional to the number of serial exchanges between parallel chains, though it surpasses the simulated annealing solution more quickly and finds the most optimal solution state. Lastly, Adaptive MCMC samples the slowest, due to our implementation that serializes each iteration of Gibbs sampling on the fast, then slow chain. With a better Adaptive MCMC implementation that parallelizes the two chains, its runtime should be closer to that of parallel tempering.

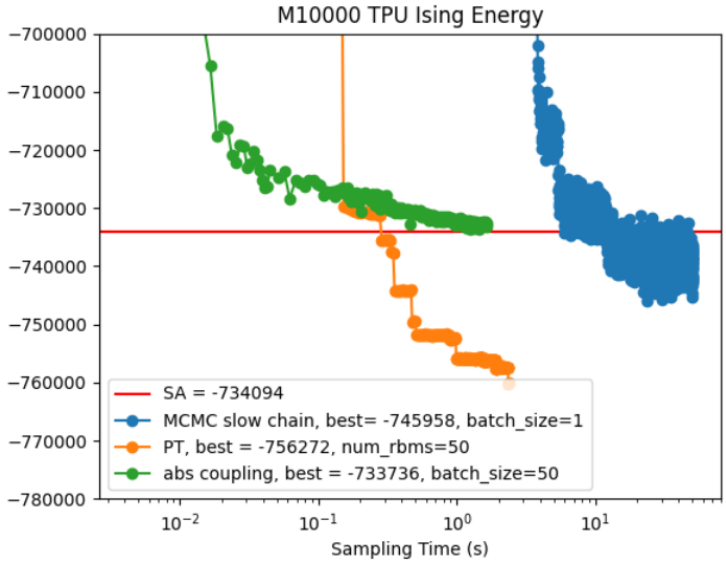

Figure 12: Runtime comparisons of RBM sampling algorithms on the TPU.

After applying parallel tempering to the M100000 problem, we are able to improve our lowest energy found from 95% to 98% of the HNN benchmark. However, we are still a ways off from the Simulated Bifurcation and Simulated Annealing results, suggesting that optimizations can be made through parameter tuning on the RBM as well as the parallel tempering algorithm.

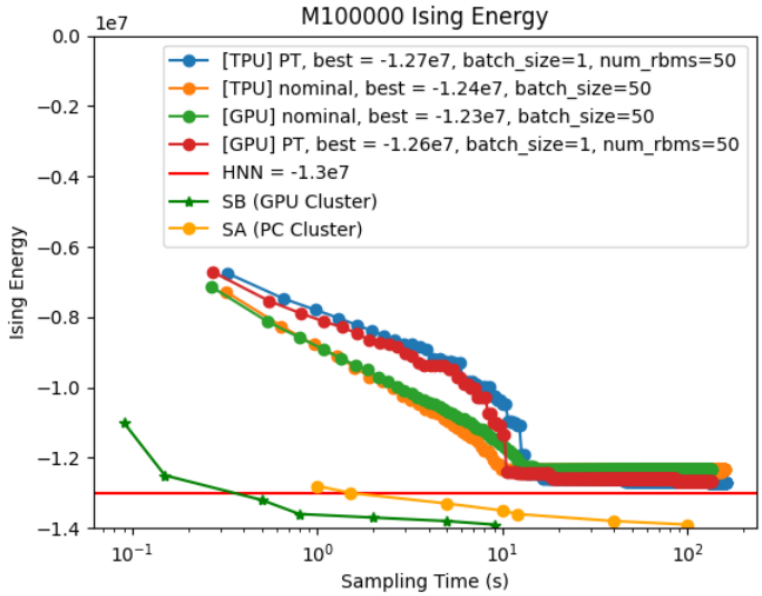

Figure 13: In these plots, 'nominal' refers to standard Gibbs sampling on the RBM optimized with absolute couplings. GPU runs finish ahead of TPU runs for reasons discussed in section 2.3.

Algorithm	Hardware	Interconnect	Total Memory	Total Peak TFLOPs	Time to HNN
Simulated Bifurcation	8 GPU Nvidia Tesla V100-SXM2-16GB	300 GB/s	128 GB	126	8ms/step * 35 steps to HNN solution = **0.28s**
Simulated Annealing	25-node PC Cluster Each Node: 2 Intel Xeon E5-2697-v3	InfiniBand (Datalink speed not specified)	128 GB per node	40	**1s** to HNN solution
RBM (standard Gibbs)	8 TPUv2 Cores	500 GB/s	64 GB	180	30ms/step * 250 steps to **90%** of HNN solution = **7.5s**
RBM (Parallel Tempering)	8 TPUv2 Cores	500 GB/s	64 GB	180	325ms/step * 500 steps to **98%** of HNN solution = **162s**
RBM (Parallel Tempering)	1 GPU Nvidia A100-80GB	N/A (All computation on one GPU)	80 GB	312	273ms/step * 500 steps to **98%** of HNN solution = **134s**

Table 4: A comparison summary of the RBM's performance on the 100,000-node Ising problem.

5 Conclusion

5.1 Future Work

In this work, we demonstrate that RBMs distributed on multiple TPUs can be used to solve the K2000 Max-Cut problem and approach optimal solutions for graphs of up to 100,000 nodes. The TPU's architecture and distributed network allow us to scale to even larger Ising problem sizes as memory resources afford.

More work can be pursued in understanding how the RBM's parameters can influence the sampling distribution and speed of convergence to the ground state, and automated parameter tuning on the device can speed up optimizations of the RBM for other graph types. Furthermore, the efficiency of the parallel tempering algorithm can be improved by removing the bottleneck of serially applying neighboring swaps.

With further scaling and improvements, RBMs on traditional hardware accelerators provide a potential method for solving combinatorial optimization problems through hardware accelerators.

5.2 Acknowledgments

Thank you to Saavan Patel, Professor Salahuddin, and the rest of the RBM subgroup for their encouragement and guidance as I learned how to research throughout my undergraduate studies up until now. Thank you to my friends and family as well for their support throughout my education!

6 References

1. Lucas, Andrew. “Ising Formulations of Many NP Problems.” Frontiers in Physics, vol. 2, 2014. Crossref, https://doi.org/10.3389/fphy.2014.00005.
2. Hayato Goto et al., Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems. Sci. Adv.5,eaav2372(2019).
3. Hayato Goto, Kotaro Endo, Masaru Suzuki, Yoshisato Sakai, Taro Kanao, Yohei Hamakawa, Ryo Hidaka, Masaya Yamasaki, Kosuke Tatsumura, High-performance combinatorial optimization based on classical mechanics, Science Advances, 7, 6, (2021).
4. Barahona, Francisco, et al. “An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design.” Operations Research, vol. 36, no. 3, 1988, pp. 493–513. JSTOR.
5. S. Patel, L. Chen, P. Canoza, S. Salahuddin, Ising model optimization problems on a FPGA accelerated restricted Boltzmann machine. arXiv: 2008.04436 (2020).
6. Yang, Kun, et al. “High Performance Monte Carlo Simulation of Ising Model on TPU Clusters.” Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Nov. 2019.
7. Kumar, Sammer, et al. "Exploring the limits of Concurrency in ML Training on Google TPUs.", arXiv: 2011.03641 (2020).
8. N. P. Jouppi et al., "Ten Lessons From Three Generations Shaped Google’s TPUv4i : Industrial Product," 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2021, pp. 1-14.
9. T. Norrie et al., "Google's Training Chips Revealed: TPUv2 and TPUv3," 2020 IEEE Hot Chips 32 Symposium (HCS), Palo Alto, CA, USA, 2020, pp. 1-70.
10. T. Norrie et al., "The Design Process for Google's Training Chips: TPUv2 and TPUv3," in IEEE Micro, vol. 41, no. 2, pp. 56-63, 1 March-April 2021.
11. Kalamkar, Dhiraj, et al. "A study of BFLOAT16 for deep learning training." arXiv preprint arXiv:1905.12322 (2019).
12. “Cloud TPU System Architecture” Google, https://cloud.google.com/tpu/docs/system-architecture-tpu-vm.
13. Fischer, A., Igel, C. (2012). An Introduction to Restricted Boltzmann Machines. In: Alvarez, L., Mejail, M., Gomez, L., Jacobo, J. (eds) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. CIARP 2012. Lecture Notes in Computer Science, vol 7441. Springer, Berlin, Heidelberg.
14. Salakhutdinov, Ruslan. “Learning Deep Boltzmann Machines using Adaptive MCMC.” International Conference on Machine Learning (2010).
15. K. Cho, T. Raiko and A. Ilin, "Parallel tempering is efficient for learning restricted Boltzmann machines," The 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 2010, pp. 1-8.
16. “Jax - the Sharp Bits.” - JAX Documentation, https://jax.readthedocs.io/en/latest/notebooks/Common_Gotchas_in_JAX.html#in-place-updates.
17. Martin Lingenheil, Robert Denschlag, Gerald Mathias, Paul Tavan, Efficiency of exchange schemes in replica exchange, Chemical Physics Letters, Volume 478, Issues 1–3, 2009, Pages 80-84, ISSN 0009-2614.